Journal of Soft Computing and Information Technology (JSCIT)
Babol Noshirvani University of Technology, Babol, Iran
Journal Homepage: jscit.nit.ac.ir
Volume 12, Number 1, Spring 2023, pp. 41-51
Received: 09/26/2020, Revised: 10/03/2021, Accepted: 01/22/2022

Checking Reachability Property in Complex Concurrent Software Systems with a Knowledge Discovery Approach

Jaafar Partabian[1], Karamollah Bagherifard[2*], Vahid Rafe[3], Hamid Parvin[4], Samad Nejatian[5]

1- Department of Computer Engineering, Lamerd Branch, Islamic Azad University, Lamerd, IRAN.
2*- Department of Computer Engineering,Yasooj Branch, Islamic Azad University, Yasooj, IRAN.
3- Department of Computer Engineering, Faculty of Engineering, Arak University, Arak 38156-8-8349, IRAN.
4-Department of Computer Engineering, Nourabad Mamasani Branch, Islamic Azad University, Nourabad Mamasani, IRAN.
5- Department of Electrical Engineering Yasooj Branch, Islamic Azad University, Yasooj, IRAN.
[1]jaafar_partabian@yahoo.com, [2*]k.bagheri@iauyasooj.ac.ir, [3]v-rafe@araku.ac.ir, [4]parvinhamid@gmail.com, [5] Samad.nej.2007@gmail.com

* Corresponding author address: Department of Computer Engineering Yasooj Branch, Islamic Azad University, Yasooj, IRAN.

Abstract- The model checking technique is a formal and effectual way in verification of software systems. By the generation and investigation of all model states, it analyses the software systems. The main issue in the model checking of the complicated systems having wide or infinite state space is the lack of memory in the generation of all states, which is referred to as "state space explosion". The Random Forest algorithm which is capable of knowledge discovery faces the above-cited problem by selecting a few promising paths. In our suggested method, first a small model of the system is developed by the formal language of graph transformation system (GTS). A training data set is created from the small state space. The generated training data set is made available to the Random Forest algorithm to detect and discover the logical relationships existing in it. Then, the knowledge acquired in this way is used in the smart and incomplete exploration of the large state space. The proposed approach is run in GROOVE which is an opensource tool for designing and studying the model of graph transformation systems. The results show that, in addition to increasing the intelligence of the model checking process, the suggested method requires less initial parameter adjustment. The proposed approach is implemented on several well-known problems. According to the experimental results, the proposed method performs better than the earlier ones in terms of average run time, the number of explored modes and the accuracy.

Keywords-Software systems verification, Knowledge discovery, State space explosion, Model checking.

I.INTRODUCTION

The need to produce error-free systems concomitant with the increasing use of software in everyday life is more and more felt. Generating secure systems is of great importance especially in critical applications where any contingent error may lead to loss of many lives. Verification process in the critical applications should be carried out in the analysis and design phase prior to the implementation stage. Among other common ways, the checking model technique is employed as a formal verification approach seeking to find out whether the model of a system meets the predetermined requirements.

Graph transformation system is widely used in modelling, and allows us to describe system behaviour and states formally in the form of graphs and charts. The model checking, though an efficient approach in discovering the system's error, would encounter the state space explosion while needing to generate the whole state space of the model. When the problem's dimensions

enlarge, the state space grows exponentially and the memory of the checking system could not produce, maintain, and deal with all states, contributing no avail to the checking process due to shortage of memory. Approaches are initiated in recent years to moderate the state space explosion. They include: Symbolic model checking[6] where the model is compressed and reduced in size using decision diagrams; Symmetry reduction[7] which merges those model states having structural similitude to make a smaller model; partial order reduction[8] which simplifies the models created by parallel ones so that by changing the execution order of the independent procedures, one could remove certain states; scenario-driven model checking[9] removes the states and transitions with no effects in the models based on graph transformation system and then sees to verification of the scenario; Abstract methods[10] merge the similar structures in the state space of the model and try to reduce the size of the latter.

All the approaches above aim to diminish the model state space and do not use any heuristics in the model checking. Notwithstanding the reduction of state space explosion by these methods, the shortage of memory and low speed yet hinder the model checking procedures in complex systems.

Although the above techniques have been almost able to tackle the problem of state space explosion, the accuracy and convergence speed of the proposed solutions are still low, especially in large and complex systems. Also, most solutions only identify deadlocks, while more complex types of properties, such as liveness and reachability, still remain. In this paper, by making intelligent the model checking technique via the supervised machine learning algorithm, it is tried to make possible the checking of the reachability feature, besides achieving a higher performance than previous methods. In the suggested method, first the necessary knowledge about applying rules (actions that change the system mode) is acquired using the machine learning algorithm. Afterwards, using the acquired knowledge, the model mode space is intelligently explored.

In this paper, with the aid of machine learning algorithm, the state space of a small model of the desired system is completely explored. Paths leading to target cases (promising paths) and those leading to non-target modes are identified and placed in the training set. The machine learning algorithm discovers and learns all the logical relationships and knowledge contained in the training set. With the help of the acquired knowledge, the promising paths are identified in the large model's mode space, and only the promising paths are explored to check the reachability property. By intelligently and incompletely exploring the problem's state space, the suggested method overcomes the problem of state space explosion, and increases the intelligence of the model checking process. Compared to the heuristic methods, this method requires less initial parameters and can detect the target state in a shorter time. Also, it generates shorter witnesses and needs to explore less number of states until reaching the target. It also enjoys good accuracy compared to the previous approaches.

The rest of the paper is organized as follows. Section 2 includes the related works. In section 3, the description of the graph transformation system, and model checking are fully described. Section 4 is a discussion about the suggested method. The implementation and practical results pertaining to the suggested method together with those of the other methods are dealt with in Section 5. Advantages and drawbacks of the proposed method are discussed in Section 6. The last section is dedicated to the conclusion.

II. RELATED WORK

Recent studies to reduce the effect of the state space explosion problem on model checking can be divided into three different groups. The first group includes general and non-heuristic approaches by which all types of properties can be checked. These approaches such as symbolic model checking[6], partial order reduction[8], symmetry reduction[7], scenario-driven model checking[9] and abstraction[10] try to reduce the size of the model's state space regardless of the type of a given property.

The second group contains approaches which employ evolutionary techniques such as GA[3], ACO[11] and PSO[4] or simple heuristic search algorithms such as A*, Iterative deepening A* (IDA*)[12] and beam search (BS)[1] to check one or two types of properties by intelligently exploring the model's state space.

The third group includes approaches that use knowledge discovery techniques such as data mining and machine learning methods. Some of these approaches, such as data mining and Bayesian networking[13, 14], machine learning[5, 15] try to check the features of the software system by discovering intelligent exploration knowledge.

Smart methods have been recently employed for model checking. A* and Iterative Deepening A*(IDA*) algorithms are used by S. Edelkamp et al.[12] to check the safety property in the system. In the present research, the Hemming distance between the current state and the goal state is regarded as the heuristic function and then implemented in HAF-APAIN tool to assess. The results indicate that this method is capable of checking the safety property by exploring fewer scenarios than SPAIN. The BDDA* approach given by S. Edelkamp and F. Reffel[16] combines the breadth-first search algorithms with A* algorithm to avoid the exploration of unnecessary states. The results show a higher efficiency of this method in comparison with the breadth-first and depth-first search algorithms. J. Maeoka et al.[17]introduced the DFHS approach for checking the safety property of the system by adding a round back option to the breadth-first search algorithm. This approach is implemented and assessed in the checker of JPF model. The functionality of this method is proved to be much better than that of the breadth-first and depth-first search methods. A heuristics is applied in A* as well as the depth-first search methods by H. C. Estler and H. Wehrheim[18], and E. Snippe[19] for checking reachability in the systems modeled by graph transformation. The heuristic function is designed on the basis of the structural similarity between the graph corresponding to the actual state and the ultimate state

graph. This shows more enhanced outcomes relative to the depth-first and breadth-first search approaches. A heuristic function is attained by S. Ziegert[20] and J. W. Elsinga[21] aimed at reducing the size of state space which can be applied in A* search, the depth-first search, and hill climbing to check the reachability property. The obtained results of this implementation indicate better efficiency compared to the previous techniques cited by H. C. Estler and H. Wehrheim[18] and E. Snippe[19]. An approach to discover the deadlock error in the reaction systems is presented by P. Godefroid and S. Khurshid[22] where, by the help of genetic algorithm, the search is directed towards the error states rather than the full state space. This is implemented and tested in VERISOFT- a search tool for the state space of the system and proved to be able to discover the error states in a shorter time in comparison with the random methods. A genetic-based way of discovering the dead end error is given by R. Yousefian et al.[3] on the basis of the graph transformation system in which each chromosome has a path of definite length in the state space. Applying mutation and crossover operators in the production of next-generation chromosomes, it aims to discover a path corresponding to the dead end state. Such a path, if any, would be reported as a counterexample. This attempt is guaranteed by the test results as to be successful in certain large models. In some huge and complex models, however, it comes to naught. X. He et al. [23], proposed a technique to determine the dead end error in the systems described in terms of graph transformation. The approach is utilized to avoid getting caught in the local optimum trap through a combination of the bird and gravitational algorithms in the direction of the state space search. The estimation results in the GROOVE tool show that the above-mentioned technique is faster and more accurate than the depth-first and breadth-first search as well as the genetic-based algorithms. Another so-called BAPSO way to recognize the dead end state in the software systems is proposed by R. Yousefian[24] which uses bat and bird algorithms. The assessment of this method in GROOVE proves that it has a more proper efficiency than each of the bat and bird algorithms though having been failed in complex systems. The ant-colony based methods are designed by E. Alba et al.[25], L. M. Duarte et al.[26], and B. L. Webster[27] towards discovering the error state in the model checking process. Since the ant algorithm is set up on the ant's quest for food in the shortest way, generation of short-length discovering paths may be conducive to less storage space for the states. Overall, these approaches could have found the optimum or near-to- optimum responses. By using the bonus-based reinforcement learning, a new approach was proffered by R. Behjati[28] which has been applied in checking of the liveness property of the on-the-fly model. The checking of the on-the-fly model, unlike the ordinary one, is carried out simultaneously with the state space exploration. The results are indicative of insufficient accuracy and the low speed of this method compared to the meta-heuristic ones. E. Pira et al.[14] give a heuristic approach based on data mining to engage in checking safety, liveness, and reachability properties in the complex software systems. By discovering repetitive patterns of a small model of the problem, they could achieve the counterexample negating the above-mentioned properties in the large model. Enjoying though more speed and accuracy relative to the heuristic approaches, it hinges upon the small model of the problem and requires setting the initial parameters in the discovering function of repetitive patterns. By dint of the machine learning technique, E. Pira et al.[13] initiated a method for refutation of safety, liveness and verify of the reachability property. Thereupon, alongside the state space exploration, the interdependencies between the rules governing the state space are extracted by the Bayesian network to the purpose of the network enhancement. Then, the discovered knowledge arisen from the Bayesian network is applied to the exploration of the rest of the state space. The GROOVE results represent a good efficiency of this method compared with the evolutionary and meta-heuristic ones. This manner depends on the selection of the part of the state space used in learning the dependencies by the help of Bayesian network. Using the machine learning technique and discovering promise paths in the small model and the exploration of the paths in the large model thereafter, J. Partabian et al.[5] have overcome the problem of state space explosion. In comparison with the evolutionary and heuristic methods, this scheme could generate the witness with a shorter length. While managing the state space in large and complex systems, M.Yasrebi et al.[15] achieved a higher accuracy in discovering dead end error via the n-gram technique. Requiring a relatively outsized space to store the n-gram table, this method is not so optimal as regards memory consumption.

Though the above-cited techniques have been able to fairly manage the issue of the state space explosion, the accuracy and speed of the convergence of the proposed methods are yet low especially in large and complex systems. Also, most approaches have only recognized the dead end while more complicated features such as reachability and liveness are still remained to be discussed. By smartening the model checking technique via the supervised machine learning algorithm, it is tried in this paper to achieve a higher efficiency than the previous methods as well as to make it possible to check the reachability property. In our suggested method, first the necessary knowledge is acquired about the operations changing the state of the system using the machine learning algorithm. Then, by the knowledge gained, the state space of the model is traversed smartly.

In exploring the state space of systems modeled by transformation of the graphs, the rule applied in the current state specifies the rule that can be executed in the next state. In other words, the law allowed in the current situation depends only on the law applied in the preceding one, not on the laws relating to the earlier cases. Based on this fact, Pira et al.[29] used the Markov chain (MC) to capture these types of dependencies and used the Estimation of Distribution Algorithm (EDA) to improve the quality of the MC. EDA is an evolutionary algorithm directing the search for the optimal solution by learning and sampling probabilistic models through the best individuals of a population at each generation. Experimental results confirm that this approach has a high speed and accuracy compared to the meta-heuristic and evolutionary methods available in the safety analysis of systems having been formally identified through graphs

transformation. In another paper Pira et al.[30] proposed a two-phase model for checking the safety of systems formally identified by graph transformation. In the first phase, the beam-search algorithm explores the state space in a certain number of states. In case of phase failure, the second phase begins: in systems specified through graph transformations, the rule applied on the previous state can determine the rule that is performed on the next state. In other words, the rule on current state depends only on the rule applied to preceding one, not the ones on earlier states. Therefore, a Markov chain (MC) is estimated to capture dependencies between the sequence of rules applied in the state space explored by the beam-search algorithm. The MC is then used to explore the remainder of the state space intelligently. To evaluate the effectiveness of the two-phase model checking, the authors implemented it in GROOVE, an opensource toolkit for designing and model checking graph transformation systems. Experimental results show that the study of the two-phase model has a high speed and accuracy in comparison to the existing meta-heuristic and evolutionary methods. Rezaei et al.[31] offers a hybrid meta-heuristic approach to cope with the problem of complete space state search of large systems. This method is employed in systems modeled through GTS. Using Artificial Bee Colony and Simulated Annealing, this approach replaces a full state space, and by producing only a portion of the system state space, checks the safety features and error (e.g., deadlock). Salimi et al.[32] in their paper present a fuzzy algorithm to analyse the reachability feature of systems modeled through GTS with large state space. To do this, the PSO algorithm is first developed to analyse the reachability property and deadlock error. Then, to increase accuracy, a fuzzy adaptive PSO algorithm is used to determine which mode and path should be checked at each step to find the target state. These two approaches are implemented in GROOVE. Experimental results indicate that the combined fuzzy approach improves the speed and accuracy compared to the other meta-heuristic algorithms such as GA and the PSO-GSA hybrid in the reachability analysis. Table1: summarizes the mentioned related works.

TABLE I: SUMMARY OF PREVIOUS APPROACHES AND SUGGESTED METHOD BASED ON THE STUDIED PARAMETERS

Parameter / Approach	Intelligence of the model checking process	Number of explored states until reaching the goal state	Runtime	Accuracy
Based on GA[3]	Low	High	Relatively High	Low
Based on PSO, PSO-GSA[4]	Low	Relatively High	Medium	Medium
Based on A*, IDA*[33]	Low	High	High	Low
Based on Datamining[13, 14]	Medium	Low	Low	Medium
Based on Machine learning[5, 15]	High	Low	Low	Relatively High
The proposed Approach (CRKD)	High	Low	Low	High

III. BACKGROUND

In the background section, the model transformation techniques and the graph transformation system are introduced and described. Model checking is one of the most important techniques of system verification that is carried out in the model design phase before the implementation operation to identify and fix possible errors. Graph transformation system is a formal method of system modelling in which the architectural components and system modes are modelled and displayed in the form of graphs.

A. Graph Transformation system

While model checking, it is necessary that the concerned system be described through a formal and intelligible modeling language.

A GTS is defined as a triple (TG, HG, R) where TG is a type graph, HG is a host graph and R is a set of graph transformation rules. The type graph TG is specified by a tuple (TGN, TGE, src, trg) in which TGN is set of node types (vertices) and TGE includes set of edge types. src and trg are two functions that assign the source and target nodes to any edges[2]. The host graph HG over TG is determined by a graph morphism type G: HG→TG that assigns a type to every node and edge in HG. In other words, the host graph should be an instance of the type graph. Also, the host graph demonstrates the initial configuration of a system. A graph transformation rule is defined as p: L→R in which L (left-hand side) and R (right-hand side) describe the pre-conditions and post-conditions of the rule, respectively. The left and right sides should conform to the type graph[5].

The rule dependence graph for a model is a directed acyclic graph (DAG) whose vertices and edges show, respectively, the rules of the model and the dependence between the rules[13]. In the software described with the graph transformation system, there exists a relationship between the rules in the paths used in the state space. For example, in the dining philosophers problem, initially only the go-hungry rule can be applied to the elementary host graph because all philosophers are in a state of thinking. After applying the go-hungry rule to the initial state, the get-left rule will be activated. Therefore, it can be inferred that the get-left rule depends on the go-hungry rule. Figure (1) shows the graph of rules dependence in the dining philosophers problem.

Fig. 1. A dependency graph for the dining philosophers problem.

The GROOVE tool has the capability of automatic checking through generating the state space of the problem. This tool is open source and features could be added to it. This is used here in modeling and analysis of software systems described by the official language of graph transformation.

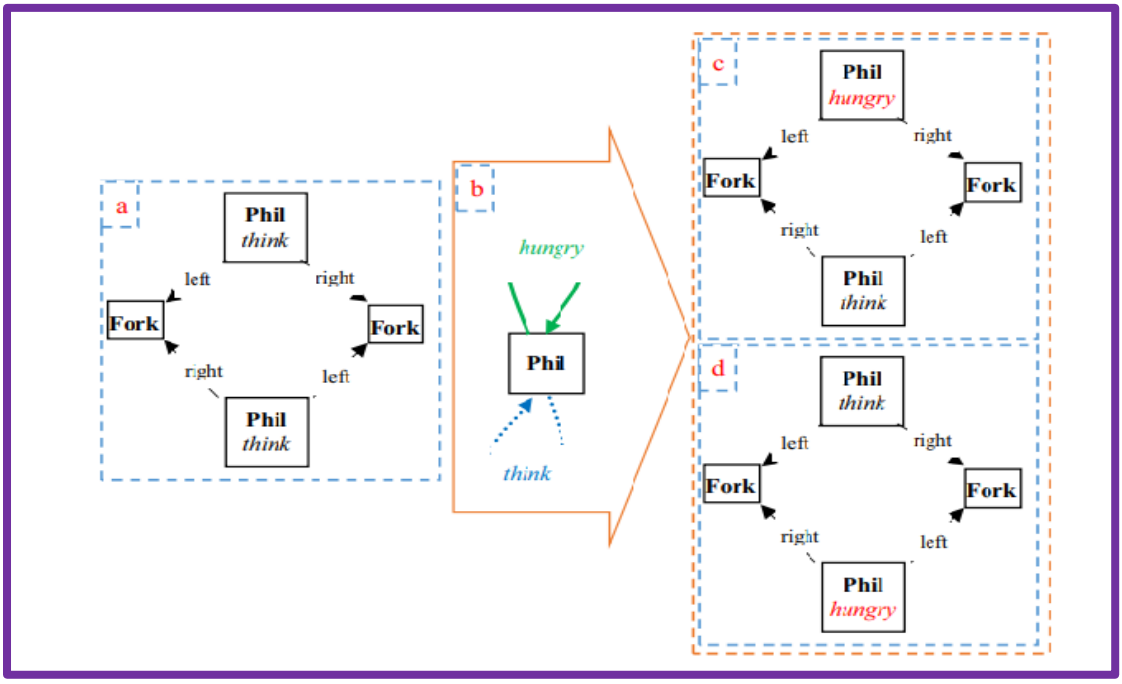

Fig. 2. Details of modeling of Dining Philosophers problem in GROOVE.

As an example of a system modelled via GROOVE, one can mention the Dining Philosophers problem with 2 philosophers (see Fig. 2). As it is shown, Fig. 2a is the host graph. Also, Fig. 2b displays the go-hungry transformation rule in which the blue double-bordered nodes and dashed edges specify the LHS graph and the fat green solid nodes and edges define the RHS graph. As shown in the figure, by applying this rule on a host graph, an edge with label think replaced by a new edge with label hungry. The results of applying this rule on the host graph of Fig. 2a are displayed in Fig. 2c, d. In GROOVE, the label of a node is specified by a self-loop edge. For example, the labels hungry and think in Fig. 2b are specified by self-loop edge.

B. Model Checking

In model checking, first all the likely cases of the model are traversed and then the correctness or incorrectness of the properties is ensured. In case the checking is done well (the state space explosion does not occur), a counterexample/witness is developed.

The counterexamples/witnesses report certain optimal /adverse behaviours of the system and could be used by the experts in fixing design flaws. Among others, safety and reachability property are important features of the software systems checked by this technique. The safety feature indicates that a good/bad item in an assumed system is absolutely true/false. Since the confirmation of this feature entails considering all the system's cases, it is tried that this property is refuted in the sense that an error is emerged in the state space. If so, the path from the beginning stage of the state space towards an error is called a counterexample. Reachability claims that there is a case in the state space in which the supposed property holds (goal state). In this case, the path from the beginning situation of the state space and ending this condition is referred to as a witness.

IV. The Proposed Approach (CRKD)

The CRKD is designed in three following phases.

A. First phase: Checking the Small Model State Space

At this phase, a small model of the system is created before the state space is traversed totally for checking the reachability property. The following operations are implemented duly.

A small model of the system in the form of the type graph is automatically or manually created by the designer.

The whole state space of the small model is made into a graph where the nodes are the states and the edges are the rules. To distinguish the goal and non-goal states, the state space is searched completely through the BFS algorithm. The state in which the reachability property holds is considered as the goal state. The BFS search is given in Algorithm 1.

Algorithm 1. The BFS algorithm in GROOVE

```
1. Input: M : a model described by GTS;
2. Output: S: the state space of M;
3. GraphState state = the initial state of M;
4. LinkedList<GraphState> stateQueue=new
LinkedList<GraphState> ();
5. stateQueue.enqueue (state);
6. S.nodes.add (state);
7. while stateQueue.size () > 0 do
8.      state = stateQueue.dequeue ();
9.      foreach MatchResult match in state.getMatches () do
10.        GraphState next = state.applyMatch(match);
11.        if next != null then
12.            if !S.nodes.contains (next) then
13.                stateQueue.enqueue (next);
14.                S.nodes.add (state);
15.                end if
16.     S.edges.add (new Transition (state,match,next));
17.     end if
18.     end for
19. end while
20. return S;
```

B. Second Phase: training and Learning

The learning set is taught to the machine learning algorithm to discover the logical relationships and the knowledge behind it. The operations below are carried out at this phase.

All the paths existing in the state space graph are extracted. Any path is generated as s0 r0 s1 r1…rl-1 sl where the si are the states and the ri represent the rules. s0 is taken as the initial state and sl is the state traversed in

Algorithm 2. The training dataset generator algorithm.

```
1. Input SM: a small model graph;
2. Output Dtr: training dataset;
3.     foreach path in SM do
4.         if reachability property has approved in the path Then
5.               set label 1;
6.          else
7.               set label 0;
8.         end if;
9.       remove all state from the path;
10.      add path into the Dtr;
11.    end foreach;
12.  return Dtr;
```

the last breadth. Removing states from each path gives a sequence of rules. Each sequence as a training tuple along

The learning set is supplied to the random forest learning algorithm to label the paths with 1 and 0 via the knowledge acquired from the small model's state space. Those labelled 1 are traversed as the promise paths in the large model's state space, and if there is any goal state, they are shown as a confirmation sign of the reachability property. Hence, the smart and incomplete exploration of the large model's state space makes keeping away from the state space explosion. The smart engaging of the large model is given in Algorithm.

D. Random Forest Learning

To enhance the accuracy, one could, among other ways, use a combination of models rather than resorting to just one model. The combination algorithms are those which take a set of models and merge their outputs to the purpose of making the ultimate learner such that its efficiency outshines that of each of the basic learners used in the algorithm. At last, the labels of the new records are determined by combining the output of each basic model used.

In the current paper, we take on the random forest combination approach. The classifications used in the random forest are all of the decision tree type. The general trend for generation of T decision trees is as with the situation of reachability property in the corresponding path is labelled in the learning set so that any path satisfying the property is labelled 1 and otherwise it is labelled 0. The generation of the learning set is demonstrated in Algorithm 2.

The learning set is provided to the machine learning algorithm to discover the rules governing the state space and to acquire the relevant knowledge (training and learning operations). Algorithm 3 represents the learning set generation.

C. Third Phase: Smart Exploration of the Large Model

With due regard to the fact that checking all the paths in the state space sparks state space explosion, we shall choose only a few paths in a smart way by the knowledge obtained from the second phase and call them as the promise paths to be taken into account in the sequel. The following operations are done in the third phase

We create the state space of the large model up to a certain level (up to the small model's level) and then we shall extract all the paths in the graph and place in the learning set. (Any path is an unlabelled test tuple

Algorithm 3. Intelligent checking of the large model.

```
1. Input: Dtr: training dataset, Dte: testing dataset, LM: large model;
2. Output: a witness for reachability property;
3. Rf= Random forest (Dtr);
4.    foreach tuple in Dts do
5.        if Rf.predict (tuple) equal 1 then
6.            explore the tuple in the LM as promising path;
7.                if current state is a goal state then
8.                    return the path as a witness;
9.                end if;
10.       endif;
11.   end foreach;
```

follows. In each iteration (t=1, 2, …, T), a learning set Di is created by means of replacement sampling. Since the replacement sampling is adopted, it may be well happened that some tuples belong in Di more than once while some others are not present in this learning set. We denote the number of the special features used in determination of branches in each tree node by m which is less than those features at hand. Among these m special features, the one with the highest information gain is chosen as the special feature of the branch. algorithm 4 shows the pseudocode of the random forest learning approach.

Algorithm 4. Random forest-to create composite mode of classification.

```
1. Input: D, a set of N class-labeled training tuples. T, the number of tree. B, the
   number of nodes;
2. Output: A composite model;
3. for t=1:T do
4.   Randomly sample the train data D with replacement to produce Di
5.   Grow on unpruned decision tree.
6.     for b=1:B do
7.       Select m variable at random from the ρ variable.
8.       Pick the best variable with the highest information gain among the m.
9.           Split the node into two daughter nodes.
10.              end for
11.          end for
12. to make a prediction at a new tuple X:
13. return majority vote the prediction of the T tree.
```

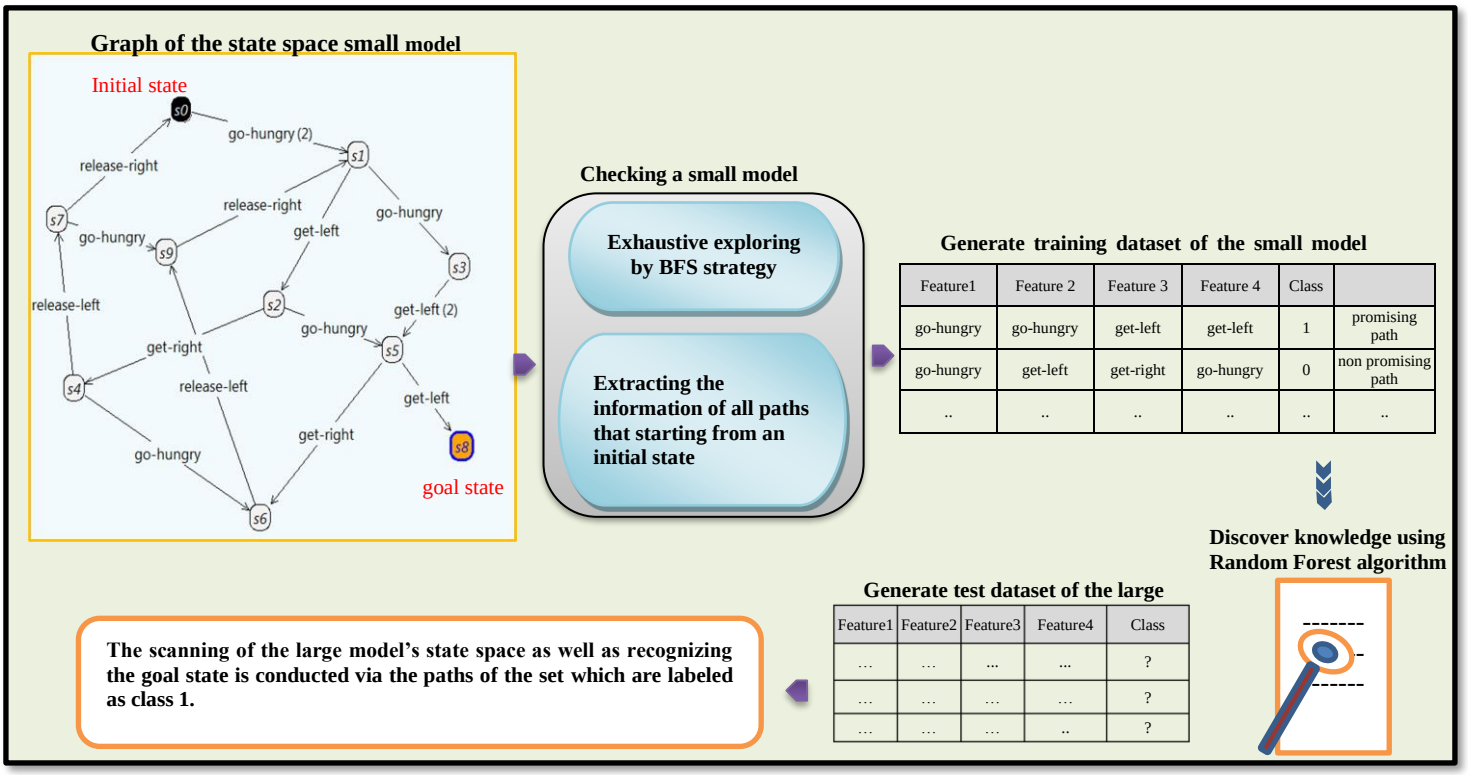

Fig. 3. Diagram of the proposed method for Dining Philosophers problem with two philosophers.

V. Experimental Result

This section is dedicated to assessing the efficiency of the proposed method for the verification of the reachability property. We have written the proposed approach (CRKD) in Java and implemented it in the opensource GROOVE to evaluate the efficiency of the method and compare it with the other techniques. The new so-called 'machine learning' strategy is added to GROOVE in which the input parameters inclusive of the model itself are determined for checking and the desired depth with the aim of exploration restriction. At the end of the implementation, one could observe the witness. Our the proposed method is assessed and compared with the approaches based on heuristic search such as: BS[1], BFS, and DFS, as well as with the meta-heuristic and evolutionary methods like GA[3], PSO, PSO-GSA[4] and the machine learning-based method[5].

A. Benchmark

The suggested method was checked and performed on four well-known problems whose checking in the graph transformation systems is impossible due to largeness of the state space. The four problems dealt with in the present paper are Dining Philosophers[34], Readers- Writers[35], N-Queens, and Process life-cycle[20].

1) Dining Philosophers

In this problem, n philosophers are sitting round a table with n forks. Each philosopher thinks, then gets hungry and first grabs the left fork and then the right fork to start eating. Since each fork is shared by two adjacent philosophers, they compete to remove the forks.

2) Readers-Writers problem

In this problem, several processes compete with each other to access simultaneously to the common sources. Some processes play the role of readers of the sources and some others need to write in them. The rule is that several readers could read a source only if no writer is writing the source. Also, at any given moment, only one writer can write in one source.

3) N- Queen problem

This problem comprises N queens in an N×N chessboard. The queens must be so arranged in the board that they cannot guard each other. Due to the movement of a queen horizontally, vertically, and obliquely in the chessboard, the queens each should be placed at different lengths, widths, and diameters to avoid be guarded.

4) Process life-cycle problem

In the life-cycle, after creating the process, should there is enough memory, it is loaded into memory and waits for CPU or I/O devices. Following the completion of the implementation, the process releases the resources at its disposal and stops.

B. Result and Analysis

The experiments are conducted in the GROOVE tool with the help of processor Intel® Core™ i5, with memory 3GB under Windows 8 Ultimate.

Table2 shows the most important parameters and their suitable values for performing all approaches.

TABLE II: THE INITIAL VALUES OF PARAMETERS

Parameter	Value	
BS[1]	dining philosophers	10
	readers-writers	80
	life-cycle	50
	N-queens	40
GA[3]	Crossover rate	50%
	Mutation rate	30%
	Position of crossover	Middle of chromosomes
PSO[4]	C1	2
	C2	2
	W	0.8

Dining Philosophers Problem

The "dining philosophers" problem uses a small model with two philosophers for learning and modelling the state space. Here, the q-state is considered as "all philosophers grab the left fork and wait for the right fork". Table 3 shows the implementation results of all methods to approve the reachability property in q-state. Empirical experiments show that all methods are capable of recognizing the goal state in this problem. The CRKD applied for larger models presents a superior efficiency than the other methods. The BS algorithm traverses the state space by both depth and surface simultaneously whence enjoying a high chance to find the state space.

Readers-Writers Problem

As for the Readers-Writers problem, two readers and two writers are taken up to make the small model presuming the q-state as "all readers/writers have ended up their processes". Owing to the fact that the state space of this problem is broad and wide, the BS algorithm takes a lengthy running time. The CRKD has a better functionality in the larger models with respect to the other approaches. Table 4 gives the outcomes of all methods for verify of the reachability property of the q-state.

N-Queen Problem

Now, we turn to the N-queen problem for which a small 4×4 model is designed. The q-state here is read "all queens are located in the right position". Table5 includes the implementation results of all approaches for verify of the reachability property of the q-state. Although only two rules govern this problem, the state space is broad and wide so that not a goal state can be recognized by the BS algorithm. The proposed approach (CRKD) takes a shorter running time especially in the larger models.

Process Life-Cycle Problem

The problem of process life-cycle works with a small model possessing 3 processes and 3 memories. The q state is assumed to be "all processes have completed their implementation". Table 6 represents the implementation of all approaches for gaining verify of the reachability.

property as regards the q state. In this problem, the goal states are deeply situated in the state space. Excluding the proposed method, all the other approaches are not able to find the goal state in the models with more than 12 processes. Prior to reach the goal state, the BS algorithm too faces lack of memory whence unsuccessful in attaining the goal state.

TABLE III: RESULTS OF ALL METHODS FOR VERIFY OF REACHABILITY PROPERTY IN DINING PHILOSOPHERS PROBLEM

Number of phiil / Methods	20 (sec)	25 (sec)	30 (sec)
BS[1]	116.63	342.54	873.23
GA[3]	12.38	27.31	75.62
PSO[4]	62.32	87.64	123.79
PSO-GSA[4]	55.28	82.84	102.94
CRKD	5.52	6.17	6.96
Model checking+ AdaBoost[5]	5.7	6.87	7.08

TABLE IV: RESULTS OF RUNNING TIME FOR ALL APPROACHES TO GET REACHABILITY PROPERTY VERIFY IN READERS-WRITERS PROBLEM

Number of R/W / Methods	4-R-4-W (sec)	5-R-5-W (sec)	6-R-6-W (sec)
BS[1]	254	492	549
GA[3]	9.72	63.3	164
PSO[4]	9.2	38	74
PSO-GSA[4]	13	19	53
CRKD	4.2	5.1	5.7
Model checking+ AdaBoost[5]	6.05	8.3	10.76

TABLE V: RESULTS OF RUNNING TIME FOR ALL APPROACHES TO GAIN REACHABILITY PROPERTY VERIFY IN N-QUEENS PROBLEM

Dimensions / Methods	8×8 (sec)	16×16 (sec)	20×20 (sec)
BS[1]	Out of Memory		
GA[3]	6.31	Out of Memory	
PSO[4]	28.19	Out of Memory	
PSO-GSA[4]	31.94	Out of Memory	
CRKD	9.58	22.4	40.08
Model checking+ AdaBoost[5]	10.73	27.3	52.41

TABLE VI: COMPARISON BETWEEN IMPLEMENTATION RESULTS TO GET REACHABILITY PROPERTY IN PROCESS LIFE-CYCLE PROBLEM

Process life cycle / Methods	10 Process-10 Memory (sec)	12 Process-12 Memory (sec)	15 Process-15 Memory (sec)
BS[1]	Out of Memory		
GA[3]	7.49	14.25	Out of Memory
PSO[4]	6.17	26.43	Out of Memory
PSO-GSA[4]	46.32	312.34	Out of Memory
CRKD	8.57	13.86	17.13
Model checking+ AdaBoost[5]	11.35	17.7	25.68

In the present paper, we consider the number of explored states in order to reach the goal state in the state space as another criterion comparing the efficiencies of the various approaches. To this purpose, we have chosen an example for each problem and the corresponding results are given in Table 7. As observed in Table 7, in most models, the suggested method requires exploring of fewer number of states.

TABLE VII: COMPARISON OF THE NUMBER OF EXPLORED STATES BY ALL METHODS FOR REACHABILITY PROPERTY

Methods / Model	GA[3]	PSO[4]	PSO-GSA[4]	BS[1]	Model checking+ AdaBoost [5]	CRKD
Dining philosophers (30 philosophers)	34542	43212	39854	8670	2854	2679
Readers-writers (5-R-5-W)	75376	17531	13931	1860	2896	2187
Process life cycle (20-Process-8-Memorys)	6543	26543	43234	21175	1532	1083
N-Queen(8×8)	4623	3632	3960	Not found	1750	1153

Comparison of maximum, minimum, average run time and depth of reaching the first goal state in the suggested method with approaches based on Bayesian optimizer[2] and the method based on machine learning[5] are given in table 8.

The accuracy of the suggested method compared to some other approaches is also shown in figure 4. According to the obtained result, the proposed method can find the goal state in less depth and time. The accuracy of the CRKD is also acceptable compared to other approaches.

TABLE VIII: COMPARISON OF THE MAXIMUM/MINIMUM/AVERAGE RUN TIME AND DEPTH OF FIRST FOUND GOAL STATE BY SOME APPROACHES

Method / Model	BOAcl2[2]		BOAcln[2]		BOActp[2]		Model checking+ Ada boost[5]		CRKD	
	maximum/minimum/ average runtime	depth of first found goal state	maximum/minimum/ average runtime	depth of first found goal state	maximum/minimum/ average runtime	depth of first found goal state	maximum/minimum/ average runtime	depth of first found goal state	maximum/minimum/ average runtime	depth of first found goal state
Dining philosophers (30 philosophers)	27.56 ±8.39	106	45.36 ±14.94	177	46.18 ±17.43	178	6.9±2.94	60	5.93±1.87	60
Readers-writers (5R-5W)	5.89 ±1.72	62	6.6 ±1.95	67	7.15 ±2.9	68	7.89±2.18	65	5.48±1.42	56
Process life cycle (15Process - 15Memorys)	1.89±0.16	45	Not found	Not found	Not found	Not found	3.69±0.71	40	2.17±1.25	42
N-Queen (16×16)	115.64± 2.02	64	Not found	Not found	Not found	Not found	68.33±3.94	57	59.86±3.73	53

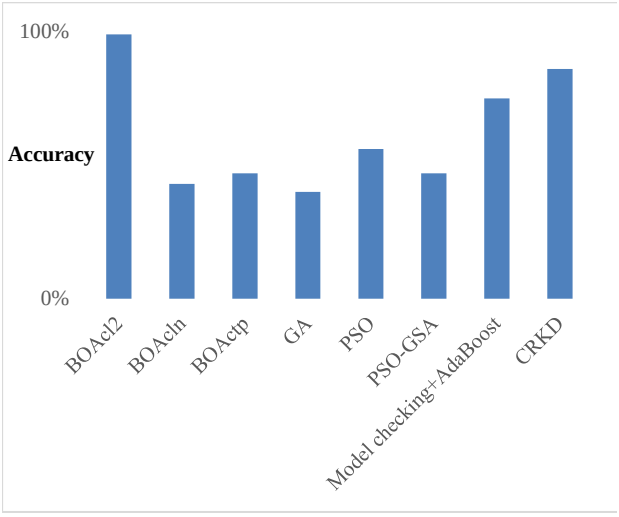

Fig. 4. Comparing the accuracy of the approaches in the dining philosopher's problem with 10 philosophers

VI. ADVANTAGES AND LIMITATIONS OF THE PROPOSED APPROACH

The proposed method has advantages in comparison with the other approaches. For instance, thanks to the usage of the machine learning algorithm, the method does the checking of the model in an incomplete and more smartly fashion to keep away from the state space explosion. To begin the checking operations, it also requires less initial parameters. The proposed approach has a higher performance speed, explores a less number of states in the state space of the model, and generates a shorter witness.

The proposed method suffers a number of limitations as well. To mention some, its accuracy depends drastically on the small model of the problem the generation of which is difficult in some problems. The efficiency of the approach is extremely reliant on the capability of the machine learning algorithm in discovering knowledge from the small model's state space. Another disadvantage could be that this technique is not able to use the small model's knowledge to manage the large model's state space in

dynamic environments where the governing conditions are frequently changing.

VII. CONCLUSION

Model checking is an automatic and appropriate way to validate the software systems. One of the most widely-used formal modelling ideas in this respect is graph transformation system. The state space explosion is the most frequent trouble which the real and complicated systems are mainly encountered. In this article, we have given an approach that could traverse the state space of the model in an incomplete and smart way and does not undergo the state space explosion. According to our proposed approach, a small model is created in the system at the outset. All the paths existing in the graph of the small model's state space are completely traversed. Those ending in the goal state are labelled 1 and the rest are marked 0 in the learning set. Then, the learning set is taught to the random forest learning algorithm in order for discovering the logical relationships between the paths and their labels. Finally, the knowledge acquired during recognizing the promise paths of the large model is used to reach the goal state. The proposed method is also made used of to check the reachability property in the large and complicated systems. It is implemented in the GROOVE tool in Java language and compared to the heuristic, meta-heuristic, and evolutionary approaches. Doing the model checking in a more smartly manner, the proposed method needs less regulated parameters and shorter average time to perform. It generates the witness with a shorter length and explores less number of states in the state space than the other approaches. Dependence on the capability of the machine learning algorithm is among the restrictions hampering the proposed approach. In systems with dynamic state space undertaking no set rules, the approach retains no suitable efficiency. The proposed method can overcome the problem of state space explosion by intelligently and incompletely exploring the state space. This method also increases the intelligence of the model checking process. The average run time is less compared to previous methods. To find the target mode, there is need to explore fewer modes. Also, this method produces shorter witnesses, and compared to the other methods, it has good accuracy.

In the future research works, other properties in the model checking technique such as liveness can be considered. Deep learning algorithms and Markov's hidden model can also be used for future work.

REFERENCES

[1] Groce, A., et al., Heuristics for model checking Java programs. International Journal on Software Tools for Technology Transfer, 2004. 6(4): p. 260-276.

[2] Pira, E., et al., Using evolutionary algorithms for reachability analysis of complex software systems specified through graph transformation. Reliability Engineering & System Safety, 2019. 191: p. 106577.

[3] Yousefian, R., et al., A heuristic solution for model checking graph transformation systems. Applied Soft Computing, 2014. 24: p. 169-180.

[4] Rafe, V., et al., A meta-heuristic solution for automated refutation of complex software systems specified through graph transformations. Applied Soft Computing, 2015. 33: p. 136-149.

[5] Partabian, J., et al., An approach based on knowledge exploration for state space management in checking reachability of complex software systems. Soft Computing, 2020. 24(10): p. 7181-7196.

[6] Zhang, H., et al. A full symbolic reachability analysis algorithm of timed automata based on BDD. in 2015 IEEE Twelfth International Symposium on Autonomous Decentralized Systems. 2015. IEEE.

[7] Lluch-Lafuente, A., Symmetry reduction and heuristic search for error detection in model checking. 2003.

[8] Lluch-Lafuente, A., et al. Partial order reduction in directed model checking. in International SPIN Workshop on Model Checking of Software. 2002. Springer.

[9] Rafe, V., Scenario-driven analysis of systems specified through graph transformations. Journal of Visual Languages & Computing, 2013. 24(2): p. 136-145.

[10] Rensink, A ,.et al. Pattern-based graph abstraction. in International Conference on Graph Transformation. 2012. Springer.

[11] Alba, E., et al. Finding safety errors with ACO. in Proceedings of the 9th annual conference on Genetic and evolutionary computation. 2007.

[12] Edelkamp, S., et al., Protocol verification with heuristic search. 2001: Bibliothek der Universität Konstanz.

[13] Pira, E., et al., Deadlock detection in complex software systems specified through graph transformation using Bayesian optimization algorithm. Journal of Systems and Software, 2017. 131: p. 181-200.

[14] Pira, E., et al., EMCDM: Efficient model checking by data mining for verification of complex software systems specified through architectural styles. Applied Soft Computing, 2016. 49 :p. 1185-1201.

[15] Yasrebi, M., et al., An efficient approach to state space management in model checking of complex software systems using machine learning techniques. Journal of Intelligent & Fuzzy Systems, 2020. 38(2): p. 1761-1773.

[16] Edelkamp, S ,.et al. OBDDs in heuristic search. in Annual Conference on Artificial Intelligence. 1998. Springer.

[17] Maeoka, J., et al., Depth-first heuristic search for software model checking, in Computer and Information Science 2015. 2016, Springer. p. 75-96.

[18] Estler, H.-C., et al. Heuristic search-based planning for graph transformation systems. in ICAPS workshop on knowledge engineering for planning and scheduling (KEPS 2011). 2011.

[19] Snippe, E. Using heuristic search to solve planning problems in GROOVE .in 14th Twente Student Conference on IT, University of Twente. Available at fmt. cs. utwente. nl/education/bachelor/73. 2011.

[20] Ziegert, S., Graph transformation planning via abstraction. arXiv preprint arXiv:1407.7933, 2014.

[21] Elsinga, J.W., On a framework for domain independent heuristics in graph transformation planning. 2016, University of Twente.

[22] Godefroid, P., et al., Exploring very large state spaces using genetic algorithms. International Journal on Software Tools for Technology Transfer, 2004. 6(2): p. 117-127.

[23] He, X., et al. A metamodel for the notation of graphical modeling languages. in 31st Annual International Computer Software and Applications Conference (COMPSAC 2007). 2007. IEEE.

[24] Yousefian, R., et al., A greedy algorithm versus metaheuristic solutions to deadlock detection in Graph Transformation Systems. Journal of Intelligent & Fuzzy Systems, 2016. 31(1): p. 137-149.

[25] Alba, E., et al. Finding deadlocks in large concurrent java programs using genetic algorithms .in Proceedings of the 10th annual conference on Genetic and evolutionary computation. 2008.

[26] Duarte, L.M., et al., Model checking the ant colony optimisation, in Distributed, parallel and biologically inspired systems. 2010, Springer. p. 221-232.

[27] Webster, B.L., Solving combinatorial optimization problems using a new algorithm based on gravitational attraction. 2004: Florida Institute of Technology.

[28] Behjati, R., et al. Bounded rational search for on-the-fly model checking of LTL properties. in International Conference on Fundamentals of Software Engineering. 2009. Springer.

[29] Pira, E., Using Markov Chain based Estimation of Distribution Algorithm for Model-based Safety Analysis of Graph Transformation. Journal of Computer Science and Technology, 2021. 36(4): p. 839-855.

[30] Pira, E., Using knowledge discovery to propose a two-phase model checking for safety analysis of graph transformations. Software Quality Journal, 2021: p. 1-28.

[31] Rezaee, N., et al., A hybrid meta-heuristic approach to cope with state space explosion in model checking technique for deadlock freeness. Journal of AI and Data Mining, 2020. 8(2): p. 189-199.

[32] Salimi, N., et al., Fuzzy particle swarm optimization algorithm (NFPSO) for reachability analysis of complex software systems. 2020.

[33] Edelkamp, S., et al., Directed explicit-state model checking in the validation of communication protocols. International journal on software tools for technology transfer, 2004. 5(2): p. 247-267.

[34] Schmidt, Á. Model checking of visual modeling languages. in CONFERENCE OF PHD STUDENTS IN COMPUTER SCIENCE. 2004.

[35] Hendrik Hausmann, J., Dynamic Meta Modeling: A Semantics Description Technique for Visual Modeling Languages. 2005.

مجله علمی پژوهشی رایانش نرم و فناوری اطلاعات
دانشگاه صنعتی نوشیروانی بابل
صفحه مجله: jscit.nit.ac.ir
جلد ۱۲، شماره ۱، بهار ۱۴۰۲، صفحه ۴۱-۵۱
دریافت: ۱۳۹۹/۰۷/۰۵، بازنگری: ۱۴۰۰/۰۷/۱۱، پذیرش: ۱۴۰۰/۱۱/۰۲

وارسی ویژگی دسترس پذیری در سیستم های نرم افزاری پیچیده و همروند با رویکرد کشف دانش

جعفر پرتابیان [۱]، کرم الله باقری فرد[۲*] وحید رافع[۳] ، حمید پروین[۴]، صمد نجاتیان [۵]،
۱- دانشکده مهندسی کامپیوتر، واحد لامرد، دانشگاه آزاد اسلامی، لامرد، ایران.
۲*- دانشکده فنی و مهندسی، واحد یاسوج، دانشگاه آزاد اسلامی، یاسوج، ایران.
۳- دانشکده فنی و مهندسی، گروه مهندسی کامپیوتر، دانشگاه اراک، اراک، ایران.
۴- دانشکده مهندسی کامپیوتر، واحد نورآباد ممسنی، دانشگاه آزاد اسلامی، نورآباد ممسنی، ایران.
۵- گروه مهندسی برق، واحد یاسوج، دانشگاه آزاد اسلامی، یاسوج، ایران.

[1]jaafar_partabian@yahoo.com,[2*]k.bagheri@iauyasooj.ac.ir,[3]v-rafe@araku.ac.ir, [4]parvinhamid@gmail.com,
[5]Samad.nej.2007@gmail.com.

* آدرس نویسنده مسوول: دانشکده فنی و مهندسی، گروه مهندسی کامپیوتر، دانشگاه اراک، اراک، ایران

چکیده -تکنیک وارسی مدل، روشی رسمی و مؤثر جهت تأیید سیستم‌های نرم‌افزاری است که با تولید و بررسی همه حالت‌های ممکنِ مدلی از سیستم نرم‌افزار به تحلیل آن می‌پردازد. چالش اساسی وارسی مدل در سیستم‌های پیچیده و بزرگ که دارای فضای حالت گسترده و یا نامحدود می‌باشند، مشکل انفجار فضای حالت (کمبود حافظه در تولید همه حالت‌های ممکن) است. الگوریتم جنگل تصادفی که قادر به کشف دانش است با انتخاب تعداد محدودی مسیر امیدبخش به مقابله با این مشکل می‌پردازد. درروش پیشنهادی، ابتدا مدل کوچکی از سیستم با استفاده از زبان رسمی سیستم توصیف گراف (GTS) ایجاد و از فضای حالت مدل کوچک، مجموعه‌ای آموزشی ایجاد می‌شود. مجموعه آموزشی تولیدشده در اختیار الگوریتم جنگل تصادفی قرار داده می‌شود تا روابط منطقی موجود در آن شناسایی و کشف شوند. سپس از دانش به‌دست‌آمده جهت پیمایش هوشمند و غیر کامل فضایِ حالتِ مدلِ بزرگ استفاده می‌شود. رویکرد پیشنهادی در ابزارGROOVE که از ابزار متن‌باز برای طراحی و بررسی مدل سیستم‌های تبدیل گراف است، اجراشده است. نتایج نشان می‌دهند که روش پیشنهادی علاوه بر افزایش هوشمندی فرایند وارسی مدل، نیاز به تنظیم پارامترهای اولیه کمتری دارد. رویکرد پیشنهادی بر روی چند مسئله شناخته شده اجرا شده است. نتایج آزمایش های تجربی نشان می‌دهند روش پیشنهادی در مقایسه با روش های قبلی متوسط زمان اجرا، تعداد حالت‌های پیمایش شده و دقت عملکرد بهتری دارد.

واژه‌های کلیدی- تأیید سیستم‌های نرم‌افزاری، کشف دانش، انفجار فضای حالت، وارسی مدل